Visibly Linear Dynamic Logic*

Alexander Weinert[1] and Martin Zimmermann[2]

1 **Reactive Systems Group, Saarland University, Saarbrücken, Germany**
weinert@react.uni-saarland.de
2 **Reactive Systems Group, Saarland University, Saarbrücken, Germany**
zimmermann@react.uni-saarland.de

Abstract

We introduce Visibly Linear Dynamic Logic (VLDL), which extends Linear Temporal Logic (LTL) by temporal operators that are guarded by visibly pushdown languages over finite words. In VLDL one can, e.g., express that a function resets a variable to its original value after its execution, even in the presence of an unbounded number of intermediate recursive calls. We prove that VLDL describes exactly the ω-visibly pushdown languages. Thus it is strictly more expressive than LTL and able to express recursive properties of programs with unbounded call stacks.

The main technical contribution of this work is a translation of VLDL into ω-visibly pushdown automata of exponential size via one-way alternating jumping automata. This translation yields exponential-time algorithms for satisfiability, validity, and model checking. We also show that visibly pushdown games with VLDL winning conditions are solvable in triply-exponential time. We prove all these problems to be complete for their respective complexity classes.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Temporal Logic, Visibly Pushdown Languages, Satisfiability, Model Checking, Infinite Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2016.28

1 Introduction

Linear Temporal Logic (LTL) [9] is widely used for the specification of non-terminating systems. Its popularity is owed to its simple syntax and intuitive semantics, as well as to the exponential compilation property, i.e., for each LTL formula there exists an equivalent Büchi automaton of exponential size. Due to the latter property, there exist algorithms for model checking in polynomial space and for solving infinite games in doubly-exponential time.

While LTL suffices to express properties of circuits and non-recursive programs with bounded memory, its application to real-life programs is hindered by its inability to express recursive properties. In fact, LTL is too weak to even express all ω-regular properties. There are several approaches to address the latter shortcoming, by augmenting LTL, for example, with regular expressions [7, 10], finite automata on infinite words [11], and right-linear grammars [13]. We concentrate on the approach of Linear Dynamic Logic (LDL) [10], which guards the globally- and eventually-operators of LTL with regular expressions. While the LTL-formula $\mathbf{F}\psi$ simply means "either now, or at some point in the future, ψ holds", the corresponding LDL operator $\langle r \rangle \psi$ means "There exists an infix matching the regular expression r starting at the current position, and ψ holds true after that infix".

* Supported by the projects "TriCS" (ZI 1516/1-1) and "AVACS" (SFB/TR 14) of the German Research Foundation (DFG).

The logic LDL captures the ω-regular languages. In spite of its greater expressive power, LDL still enjoys the exponential compilation property, hence there exist algorithms for model checking and solving infinite games in polynomial space and doubly-exponential time, respectively.

While the expressive power of LDL is sufficient for many specifications, it is still not able to reason about recursive properties of systems. In order to address this shortcoming, we replace the regular languages guarding the temporal operators with visibly pushdown languages (VPLs) [2]. We consider VPLs specified by visibly pushdown automata (VPAs) [2] in this work.

A VPA is a pushdown automaton that operates over a fixed partition of the input alphabet into calls, returns and local actions. In contrast to traditional pushdown automata, VPAs may only push symbols onto the stack when reading calls and may only pop symbols off the stack when reading returns. Moreover, they may not even inspect the topmost symbol of the stack when not reading returns. Thus, the height of the stack after reading a word is known in advance for all VPAs using the same partition of the input alphabet. Due to this, VPAs are closed under union and intersection, as well as complementation. The class of languages accepted by VPAs is known as visibly pushdown languages.

The class of such languages over infinite words, i.e., ω-visibly pushdown languages, are known to allow for the specification of many important properties in program verification such as "there are infinitely many positions at which at most two functions are active", which expresses repeated returns to a main-loop, or "every time the program enters a module m while p holds true, p holds true upon exiting m" [2]. The extension of VPAs to their variant operating on infinite words is, however, not well-suited to the specification of such properties in practice, as Boolean operations on such automata do not preserve the logical structure of the original automata. By guarding its temporal operators with VPAs, VLDL allows for modular specification of recursive properties while capturing ω-VPAs.

1.1 Our contributions

We introduce VLDL and study its expressiveness and algorithmic properties.

Firstly, we provide translations from VLDL to VPAs over infinite words, so-called ω-VPAs, and vice versa. For the direction from logic to automata we translate VLDL formulas into one-way alternating jumping automata (1-AJA), which are known to be translatable into ω-VPAs of exponential size due to Bozzelli [4]. For the direction from automata to logic we use a translation of ω-VPAs into deterministic parity stair automata (PSA) by Löding et al. [8], which we then translate into VLDL formulas.

Secondly, we prove the satisfiability problem and the validity problem for VLDL to be ExpTime-complete. Membership in ExpTime follows from the previously mentioned constructions, while we show ExpTime-hardness of the problems by a reduction from the word problem for polynomially space-bounded alternating Turing machines adapting a similar reduction by Bouajjani et al. [3].

As a third result, we show that model checking visibly pushdown systems against VLDL specifications is ExpTime-complete as well. Membership in ExpTime follows from ExpTime-membership of the model checking problem for 1-AJAs against visibly pushdown systems. ExpTime-hardness follows from ExpTime-hardness of the validity problem for VLDL.

Finally, solving visibly pushdown games with VLDL winning conditions is proven to be 3ExpTime-complete. Membership in 3ExpTime follows from the exponential translation of VLDL formulas into ω-VPAs and the membership of solving pushdown games against ω-VPA winning conditions in 2ExpTime due to Löding et al. [8]. 3ExpTime-hardness is

due to a reduction from solving pushdown games against LTL specifications, again due to Löding et al. [8].

Our results show that VLDL allows for the concise specification of important properties in a logic with intuitive semantics. In the case of satisfiability and model checking, the complexity jumps from PSpace-completeness for LDL to ExpTime-completeness. For solving infinite games, we gain an exponent moving from 2ExpTime-completeness to 3ExpTime-completeness.

We choose VPAs for the specification of guards in order to simplify arguing about the expressive power of VLDL. In order to simplify the modeling of ω-VPLs, other formalisms that capture VPLs over finite words may be used. We discuss one such formalism in the conclusion.

All proofs omitted due to space restrictions can be found in the full version [12].

1.2 Related Work

The need for specification languages able to express recursive properties has been identified before and there exist other approaches to using visibly pushdown languages over infinite words for specifications, most notably CaRet [1], and, more recently, VLTL [5]. While VLTL captures the class of ω-visibly pushdown languages, CaRet captures only a strict subset of it. For both logics there exist exponential translations into ω-VPAs. In this work, we provide exponential translations from VLDL to ω-VPAs and vice versa. Hence, CaRet is strictly less powerful than VLDL, but every CaRet formula can be translated into an equivalent VLDL formula, albeit with a doubly-exponential blowup. Similarly, every VLTL formula can be translated into an equivalent VLDL formula and vice versa, with doubly-exponential blowup in both directions.

In contrast to VLTL, which introduces substitution operators to regular expressions (replacing occurrences of local actions by well-matched words), VLDL instead extends the concepts introduced for LTL and LDL with visibly pushdown automata. Hence, specifications written in VLDL are modular and have an intuitive semantics, in particular for practitioners already used to LTL.

Other logical characterizations of visibly pushdown languages include characterizations by a fixed-point logic [4] and by monadic second order logic augmented with a binary matching predicate (MSO_μ) [2]. Even though these logics also capture the class of visibly pushdown languages, they feature neither an intuitive syntax nor intuitive semantics and thus are less applicable than VLDL in a practical setting.

2 Preliminaries

In this section we introduce the basic notions used in the remainder of this work. A pushdown alphabet $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$ is a finite set Σ that is partitioned into calls Σ_c, returns Σ_r and local actions Σ_l. We write $w = w_0 \cdots w_n$ and $\alpha = \alpha_0\alpha_1\alpha_2 \cdots$ for finite and infinite words, respectively. The stack height $sh(w)$ reached after reading w is defined inductively as $sh(\varepsilon) = 0$, $sh(wc) = sh(w) + 1$ for $c \in \Sigma_c$, $sh(wr) = \max\{0, sh(w) - 1\}$ for $r \in \Sigma_r$, and $sh(wl) = sh(w)$ for $l \in \Sigma_l$. A call $c \in \Sigma_c$ at some position k of a word w is matched if there exists a $k' > k$ with $w_{k'} \in \Sigma_r$ and $sh(w_0 \cdots w_k) - 1 = sh(w_0 \cdots w_{k'})$. The return at the smallest such position k' is the matching return of c. We define $steps(\alpha) := \{k \in \mathbb{N} \mid \forall k' \geq k.\ sh(\alpha_0 \cdots \alpha_{k'}) \geq sh(\alpha_0 \cdots \alpha_k)\}$ as the positions reaching a lower bound on the stack height. Note that $0 \in steps(\alpha)$ and that $steps(\alpha)$ is infinite for infinite words α.

Visibly Pushdown Systems. A visibly pushdown system (VPS) $\mathcal{S} = (Q, \widetilde{\Sigma}, \Gamma, \Delta)$ consists of a finite set Q of states, a pushdown alphabet $\widetilde{\Sigma}$, a stack alphabet Γ, which contains a stack-bottom marker $\bot$, and a transition relation

$$\Delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\bot\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_l \times Q).$$

A configuration (q, γ) of $\mathcal{S}$ is a pair of a state $q \in Q$ and a stack content $\gamma \in \Gamma_c = (\Gamma \setminus \{\bot\})^* \cdot \bot$. The VPS $\mathcal{S}$ induces the configuration graph $G_{\mathcal{S}} = (Q \times \Gamma_c, E)$ with $E \subseteq ((Q \times \Gamma_c) \times \Sigma \times (Q \times \Gamma_c))$ and $((q, \gamma), a, (q', \gamma')) \in E$ if, and only if, either *(i)* $a \in \Sigma_c$, $(q, a, q', A) \in \Delta$, and $A\gamma = \gamma'$, *(ii)* $a \in \Sigma_r$, $(q, a, \bot, q') \in \Delta$, and $\gamma = \gamma' = \bot$, *(iii)* $a \in \Sigma_r$, $(q, a, A, q') \in \Delta$, $A \neq \bot$, and $\gamma = A\gamma'$, or *(iv)* $a \in \Sigma_l$, $(q, a, q') \in \Delta$, and $\gamma = \gamma'$. For an edge $e = ((q, \gamma), a, (q', \gamma'))$, a is the label of e. A run $\pi = (q_0, \gamma_0) \cdots (q_n, \gamma_n)$ of $\mathcal{S}$ on $w = w_0 \cdots w_{n-1}$ is a sequence of configurations where $((q_i, \gamma_i), w_i, (q_{i+1}, \gamma_{i+1})) \in E$ in $G_{\mathcal{S}}$ for all $i \in [0; n-1]$.

The VPS $\mathcal{S}$ is deterministic if for each vertex (q, γ) in $G_{\mathcal{S}}$ and each $a \in \Sigma$ there exists at most one outgoing a-labeled edge from (q, γ). In figures, we write $\downarrow A$, $\uparrow A$ and $\rightarrow$ to denote pushing and popping A onto and off the stack, and local actions, respectively.

(Büchi) Visibly Pushdown Automata. A visibly pushdown automaton (VPA) [2] is a six-tuple $\mathfrak{A} = (Q, \widetilde{\Sigma}, \Gamma, \Delta, I, F)$, where $(Q, \widetilde{\Sigma}, \Gamma, \Delta)$ is a VPS and $I, F \subseteq Q$ are sets of initial and final states. A run $(q_0, \gamma_0)(q_1, \gamma_1)(q_2, \gamma_2) \cdots$ of $\mathfrak{A}$ is initial if $(q_0, \gamma_0) = (q_I, \bot)$ for some $q_I \in I$. A finite run $\pi = (q_0, \gamma_0) \cdots (q_n, \gamma_n)$ is accepting if $q_n \in F$. A Büchi VPA (BVPA) is syntactically identical to a VPA, but we only consider runs over infinite words. An infinite run is Büchi-accepting if it visits states in F infinitely often. A (B)VPA $\mathfrak{A}$ accepts a word w (an infinite word α) if there exists an initial (Büchi-)accepting run of $\mathfrak{A}$ on w (α). The family of languages accepted by (B)VPA is denoted by (ω-)VPL.

3 Visibly Linear Dynamic Logic

We fix a finite set P of atomic propositions and a partition $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$ of 2^P throughout this work. The syntax of VLDL is defined by the grammar

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle \mathfrak{A} \rangle \varphi \mid [\mathfrak{A}]\varphi,$$

where $p \in P$ and $\mathfrak{A}$ ranges over testing visibly pushdown automata (TVPA) over $\widetilde{\Sigma}$. A TVPA $\mathfrak{A} = (Q, \widetilde{\Sigma}, \Gamma, \Delta, I, F, t)$ consists of a VPA $(Q, \widetilde{\Sigma}, \Gamma, \Delta, I, F)$ and a partial function t mapping states to VLDL formulas over $\widetilde{\Sigma}$.[1] Such an automaton accepts an infix $\alpha_i \cdots \alpha_j$ of an infinite word $\alpha_0\alpha_1\alpha_2 \cdots$ if the embedded VPA has an initial accepting run $(q_i, \gamma_i) \cdots (q_{j+1}, \gamma_{j+1})$ on $\alpha_i \cdots \alpha_j$ such that, if q_{i+k} is marked with φ by t, then $\alpha_{i+k}\alpha_{i+k+1}\alpha_{i+k+2} \cdots$ satisfies φ.

We define the size of φ as the sum of the number of subformulas (including those contained as tests in automata and their subformulas) and of the numbers of states of the automata contained in φ. As shorthands, we use $\mathtt{tt} := p \vee \neg p$ and $\mathtt{ff} := p \wedge \neg p$ for some atomic proposition p. Even though the testing function t is defined as a partial function, we generally assume it is total by setting $t \colon q \mapsto \mathtt{tt}$ if $q \notin \mathit{domain}(t)$.

Let $\alpha = \alpha_0\alpha_1\alpha_2 \cdots$ be an infinite word over 2^P and let $k \in \mathbb{N}$ be a position in α. We define the semantics of VLDL inductively via

[1] Obviously, there are some restrictions on the nesting of tests into automata. More formally, we require the subformula relation to be acyclic as usual.

- $(\alpha, k) \models p$ if, and only if, $p \in \alpha_k$,
- $(\alpha, k) \models \neg\varphi$ if, and only if, $(\alpha, k) \not\models \varphi$,
- $(\alpha, k) \models \varphi_0 \wedge \varphi_1$ if, and only if, $(\alpha, k) \models \varphi_0$ and $(\alpha, k) \models \varphi_1$, and dually for $\varphi_0 \vee \varphi_1$,
- $(\alpha, k) \models \langle\mathfrak{A}\rangle\varphi$ if, and only if, there exists $l \geq k$ s.t. $(k, l) \in \mathcal{R}_{\mathfrak{A}}(\alpha)$ and $(\alpha, l) \models \varphi$,
- $(\alpha, k) \models [\mathfrak{A}]\varphi$ if, and only if, for all $l \geq k$, $(k, l) \in \mathcal{R}_{\mathfrak{A}}(\alpha)$ implies $(\alpha, l) \models \varphi$,

where $\mathcal{R}_{\mathfrak{A}}(\alpha)$ contains all (k, l) such that $\mathfrak{A}$ accepts $\alpha_k \cdots \alpha_{l-1}$. Formally, we define

$$\mathcal{R}_{\mathfrak{A}}(\alpha) := \{(k, l) \in \mathbb{N} \times \mathbb{N} \mid \exists \text{ init. acc. run } (q_k, \sigma_k) \cdots (q_l, \sigma_l) \text{ of } \mathfrak{A} \text{ on } \alpha_k \cdots \alpha_{l-1} \\ \textbf{and } \forall m \in \{k, \ldots, l\}.\ (\alpha, m) \models t(q_m)\}.$$

We write $\alpha \models \varphi$ as a shorthand for $(\alpha, 0) \models \varphi$ and say that α is a model of φ in this case. The language of φ is defined as $L(\varphi) := \{\alpha \in (2^P)^\omega \mid \alpha \models \varphi\}$. As usual, disjunction and conjunction are dual, as well as the $\langle\mathfrak{A}\rangle$-operator and the $[\mathfrak{A}]$-operator, which can be dualized using De Morgan's law and the logical identity $[\mathfrak{A}]\varphi \equiv \neg\langle\mathfrak{A}\rangle\neg\varphi$, respectively. Note that the latter identity only dualizes the temporal operator, but does not require complementation of the automaton guarding the operator. We additionally allow the use of derived boolean operators such as $\rightarrow$ and $\leftrightarrow$, as they can easily be reduced to the basic operators $\wedge$, $\vee$ and $\neg$.

The logic VLDL combines the expressive power of visibly pushdown automata with the intuitive temporal operators of LDL. Thus, it allows for concise and intuitive specifications of many important properties in program verification [2]. In particular, VLDL allows for the specification of recursive properties, which makes it more expressive than both LDL [10] and LTL [9]. In fact, we can embed LDL in VLDL in linear time.

▶ **Lemma 1.** *For any LDL formula ψ over P we can effectively construct a VLDL formula φ over $\widetilde{\Sigma} := (\emptyset, \emptyset, 2^P)$ in linear time such that $L(\psi) = L(\varphi)$.*

Proof. We define φ by structural induction over ψ. The only interesting case is $\psi = \langle r\rangle\psi'$, since all other cases follow from closure properties and duality. We obtain the VLDL formula φ' over $\widetilde{\Sigma}$ equivalent to ψ' by induction and construct the finite automaton $\mathfrak{A}_r$ from r using the construction from [6]. The automaton $\mathfrak{A}_r$ contains tests, but is not equipped with a stack. Since $\widetilde{\Sigma} = (\emptyset, \emptyset, 2^P)$, we can interpret $\mathfrak{A}_r$ as a TVPA without changing the language it recognizes. We call the TVPA $\mathfrak{A}'_r$ and define $\varphi = \langle\mathfrak{A}'_r\rangle\varphi'$. ◀

Since LTL can be in turn embedded in LDL in linear time, Lemma 1 directly implies the embeddability of LTL in VLDL in linear time. Note that this proof motivates the use of TVPAs instead of VPAs without tests as guards in order to obtain a concise formalism. We later show that removing tests from these automata does not change the expressiveness of VLDL. It is, however, open whether it is possible to translate even LTL formulas into VLDL formulas without tests in polynomial time.

▶ **Example 2.** Assume that we have a program that may call some module m and has the observable atomic propositions $P := \{c, r, p, q\}$, where c and r denote calls to and returns from m, and p and q are arbitrary propositions.

We now construct a formula that describes the condition "If p holds true immediately after entering m, it shall hold immediately after the corresponding return from m as well" [1]. For the sake of readability, we assume that the program never emits both c and r in the same step. Moreover, we assume that the program emits at least one atomic proposition in each step. Since we want to count the calls and returns occurring in the program using the stack, we pick the pushdown alphabet $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$ such that $P' \subseteq P$ is in Σ_c if $c \in P'$, $P' \in \Sigma_r$ if $r \in P'$, but $c \notin P'$, and $P' \in \Sigma_l$ otherwise.

Figure 1 The automata $\mathfrak{A}_c$ and $\mathfrak{A}_r$ for Example 2.

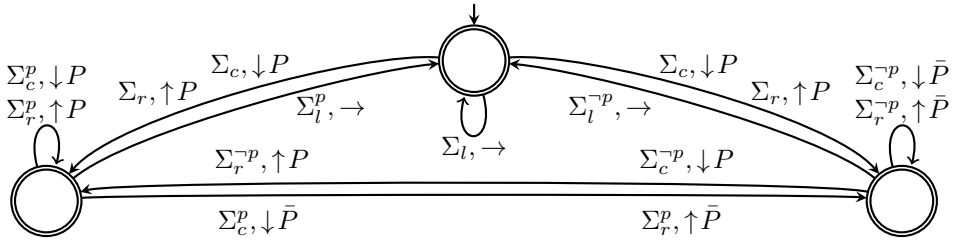

Figure 2 A BVPA $\mathfrak{A}$ specifying the same language as φ from Example 2.

The formula $\varphi := [\mathfrak{A}_c](p \rightarrow \langle \mathfrak{A}_r \rangle p)$ then captures the condition, with $\mathfrak{A}_c$ and $\mathfrak{A}_r$ as shown in Figure 1. The automaton $\mathfrak{A}_c$ accepts all finite words ending with a call to m, whereas the automaton $\mathfrak{A}_r$ accepts all words ending with a single unmatched return.

Figure 2 shows a BVPA $\mathfrak{A}$ describing the same specification as φ. Here, we use $\Sigma_x^p = \{P' \in \Sigma_x \mid p \in P'\}$ and $\Sigma_x^{\neg p} = \{P' \in \Sigma_x \mid p \notin P'\}$ for $x \in \{c, r, l\}$. In contrast to φ, which uses only a single stack symbol, namely A, the BVPA $\mathfrak{A}$ has to rely on the two stack symbols P and $\bar{P}$ to track whether or not p held true after entering the module m. Moreover, there is no direct correlation between the logical structure of the specification and the structure of the BVPA, which exemplifies the difficulty of maintaining specifications given as BVPAs.

In order to abstain from using automata, it would also be possible to formalize the specification using a VLTL formula [5] that describes the same language as φ. One such formula would be $\psi := (\alpha; \mathtt{tt}) | \alpha \rangle \mathtt{ff}$, where the visibly rational expression α is defined as

$$\alpha := [(p \cup q)^* c\, [(q\Box) \cup (p\Box p)]\, r (p \cup q)^*]^{\circlearrowleft \Box} \curvearrowleft_{\Box} (p \cup q)^*$$

that uses the additional local action $\Box$. Again, the conditional nature of the specification is lost in the translation to VLTL. Moreover, the temporal nature is not well visible in the formal specification due to use of the non-standard future weak power operator $\psi | \alpha \rangle \psi$.

In contrast to these two alternative formal specifications, VLDL offers a readable and intuitive formalism that combines the well-known standard acceptors for visibly pushdown languages with guarded versions of the widely used temporal operators of LTL and the readability of classical logical operators.

4 VLDL Captures ω-VPL

In this section we show that VLDL characterizes ω-VPL. Recall that a language is in ω-VPL if, and only if, there exists a BVPA recognizing it. We provide effective constructions for transforming BVPAs into equivalent VLDL formulas and vice versa.

▶ **Theorem 3.** *For any language of infinite words $L \subseteq \Sigma^\omega$ there exists a BVPA $\mathfrak{A}$ with $L(\mathfrak{A}) = L$ if, and only if, there exists a VLDL formula φ with $L(\varphi) = L$. There exist effective translations for both directions.*

In Section 4.1 we show the construction of VLDL formulas from BVPAs via deterministic parity stair automata. In Section 4.2 we construct one-way alternating jumping automata from VLDL formulas. These automata are known to be translatable into equivalent BVPAs. Both constructions incur an exponential blowup in size. In the construction of BVPAs from VLDL formulas, this blowup is shown to be unavoidable. It remains open whether the blowup can be avoided in the construction for the other direction.

4.1 From Stair Automata to VLDL

In this section we construct a VLDL formula of exponential size that is equivalent to a given BVPA $\mathfrak{A}$. To this end, we first transform $\mathfrak{A}$ into an equivalent deterministic parity stair automaton (DPSA) [8] in order to simplify the translation. A PSA $\mathfrak{A} = (Q, \widetilde{\Sigma}, \Gamma, \Delta, I, \Omega)$ consists of a VPS $\mathcal{S} = (Q, \widetilde{\Sigma}, \Gamma, \Delta)$, a set of initial states I, and a coloring $\Omega\colon Q \to \mathbb{N}$. The automaton $\mathfrak{A}$ is deterministic if $\mathcal{S}$ is deterministic and $|I| = 1$.

A run ρ of $\mathfrak{A}$ on a word α is a run of the VPS $\mathcal{S}$ on α. Recall that a step is a position at which the stack height reaches a lower bound for the remainder of the word. A stair automaton only evaluates the parity condition at the steps of the word. Formally, a run $\rho_\alpha = (q_0, \sigma_0)(q_1, \sigma_1)(q_2, \sigma_2)\cdots$ on the word α induces a sequence of colors $\Omega(\rho_\alpha) := \Omega(q_{k_0})\Omega(q_{k_1})\Omega(q_{k_2})\cdots$, where $k_0 < k_1 < k_2 \cdots$ is the ordered enumeration of the steps of α. A DPSA $\mathfrak{A}$ accepts an infinite word α if there exists an initial run ρ of $\mathfrak{A}$ on α such that the largest color appearing infinitely often in $\Omega(\rho)$ is even. The language $L(\mathfrak{A})$ of a parity stair automaton $\mathfrak{A}$ is the set of all words α that are accepted by $\mathfrak{A}$.

▶ **Lemma 4** ([8]). *For every BVPA $\mathfrak{A}$ there exists an effectively constructible equivalent DPSA $\mathfrak{A}_{st}$ with $|\mathfrak{A}_{st}| \in \mathcal{O}(2^{|\mathfrak{A}|})$.*

Since the stair automaton $\mathfrak{A}_{st}$ equivalent to a BVPA $\mathfrak{A}$ is deterministic, the acceptance condition collapses to the requirement that the unique run of $\mathfrak{A}_{st}$ on α must be accepting. Another important observation is that every time $\mathfrak{A}_{st}$ reaches a step of α, the stack may be cleared. Since the topmost element of the stack will never be popped after reaching a step, and since VPAs cannot inspect the top of the stack, neither this symbol, nor the ones below it have any influence on the remainder of the run.

Thus, the formula equivalent to $\mathfrak{A}_{st}$ has to specify the following constraints: There must exist some state q of even color such that the stair automaton visits q at a step, afterwards the automaton may never visit a higher color again at a step, and each visit to q at a step must be followed by another visit to q at a step. All of these conditions can be specified by VLDL formulas in a straightforward way, since $\mathfrak{A}_{st}$ is deterministic and since there is only a finite number of colors in $\mathfrak{A}_{st}$.

▶ **Lemma 5.** *For each DPSA $\mathfrak{A}$ there exists an effectively constructible equivalent VLDL formula $\varphi_{\mathfrak{A}}$ with $|\varphi_{\mathfrak{A}}| \in \mathcal{O}(|\mathfrak{A}|^2)$.*

Proof. We first construct a formula φ_{st} such that $(\alpha, k) \models \varphi_{st}$ if, and only if, $k \in \mathit{steps}(\alpha)$: Let $\mathfrak{A}_{st}$ be a VPA that accepts upon reading an unmatched return, constructed similarly to $\mathfrak{A}_r$ from Example 2. Then we can define $\varphi_{st} := [\mathfrak{A}_{st}]\mathtt{ff}$, i.e., we demand that the stack height never drops below the current level by disallowing $\mathfrak{A}_{st}$ to ever accept.

In the remainder of this proof, we write ${}_{I'}\mathfrak{A}_{F'}$ to denote the TVPA that we obtain from combining the VPS of $\mathfrak{A}$ with the sets I' and F' of initial and final states. Additionally, we require that ${}_{I'}\mathfrak{A}_{F'}$ does not accept the empty word. This is trivially true if the intersection of I' and F' is empty, and easily achieved by adding a new initial state if it is not. Furthermore, we define $Q_{even} := \{q \in Q \mid \Omega(q) \text{ is even}\}$ and $Q_{>q} := \{q' \in Q \mid \Omega(q') > \Omega(q)\}$.

Recall that $\mathfrak{A}$ accepts a word α if the largest color seen infinitely often at a step during the run of $\mathfrak{A}$ on α is even. This is equivalent to the existence of a state q as described above. These conditions are formalized as

$$\varphi_1(q) := \langle {}_I\mathfrak{A}_{\{q\}} \rangle (\varphi_{st} \wedge [{}_{\{q\}}\mathfrak{A}_{Q_{>q}}] \neg \varphi_{st})$$

and

$$\varphi_2(q) := [{}_I\mathfrak{A}_{\{q\}}](\varphi_{st} \rightarrow \langle {}_{\{q\}}\mathfrak{A}_{\{q\}} \rangle \varphi_{st}),$$

respectively. We obtain $\varphi_{\mathfrak{A}} := \bigvee_{q \in Q_{even}} (\varphi_1(q) \wedge \varphi_2(q))$.

The construction of $\varphi_2(q)$ relies heavily on the determinism of the DPSA $\mathfrak{A}$. If $\mathfrak{A}$ were not deterministic, the universal quantification over all runs ending in q at a step would also capture eventually rejecting partial runs. Since there only exists a single run of $\mathfrak{A}$ on the input word, however, $\varphi_{\mathfrak{A}}$ has the intended meaning. Furthermore, both $\varphi_1(q)$ and $\varphi_2(q)$ use the observation that we are able to clear the stack every time that we reach a step. Thus, although the stack contents are not carried over between the different automata, the concatenation of the automata does not change the resulting run. Hence, we have $\alpha \in L(\mathfrak{A})$ if, and only if, $(\alpha, 0) \models \varphi_{\mathfrak{A}}$ and thus $L(\mathfrak{A}) = L(\varphi_{\mathfrak{A}})$. ◀

Combining Lemmas 4 and 5 yields that VLDL is at least as expressive as BVPA. The construction inherits an exponential blowup from the construction of DPSAs from BVPAs. This shows one direction of Theorem 3.

In the next section we show that each VLDL formula φ can be transformed into an equivalent VPA with exponential blowup. Thus, the construction from the proof of Lemma 5 yields a normal form for VLDL formulas. In particular, formulas in this normal form only use temporal operators up to nesting depth three.

▶ **Proposition 6.** *Let φ be a VLDL formula. There exists an equivalent formula $\varphi' = \bigvee_{i=1}^{n} (\langle \mathfrak{A}_{i,1} \rangle (\varphi_{st} \wedge [\mathfrak{A}_{i,2}] \neg \varphi_{st}) \wedge [\mathfrak{A}_{i,1}](\varphi_{st} \rightarrow \langle \mathfrak{A}_{i,3} \rangle \varphi_{st}))$, for some n that is doubly-exponential in $|\varphi|$, where all $\mathfrak{A}_{i,j}$ share the same underlying VPS, φ_{st} is fixed over all φ, and neither $\mathfrak{A}_{i,j}$ nor φ_{st} contain tests.*

Proposition 6 shows that tests are syntactic sugar but removing them incurs a doubly-exponential blowup. It remains open whether this blowup can be avoided.

4.2 From VLDL to 1-AJA

We now construct, for a given VLDL formula φ, an equivalent BVPA $\mathfrak{A}_\varphi$. A direct construction would incur a non-elementary blowup due to the unavoidable exponential blowup of complementing BVPAs. Moreover, it would be difficult to handle runs of the VPAs over finite words and their embedded tests, which run in parallel. Thus, we extend a construction from [6], where a similar challenge was addressed using alternating automata. Instead of alternating visibly pushdown automata, however, we use one-way alternating jumping automata (1-AJA) [4], which can be translated into equivalent BVPAs of exponential size.

A 1-AJA $\mathfrak{A} = (Q, \widetilde{\Sigma}, \delta, I, \Omega)$ consists of a finite state set Q, a visibly pushdown alphabet $\widetilde{\Sigma}$, a set $I \subseteq Q$ of initial states, a transition function $\delta \colon Q \times \Sigma \to \mathcal{B}^+(\mathit{Comms}_Q)$, with $\mathit{Comms}_Q := \{\rightarrow, \rightarrow_a\} \times Q \times Q$, where $\mathcal{B}^+(\mathit{Comms}_Q)$ denotes the set of positive Boolean formulas over Comms_Q, and a coloring $\Omega \colon Q \to \mathbb{N}$. We define $|\mathfrak{A}| = |Q|$. Intuitively, when the automaton is in state q at position k of the word $\alpha = \alpha_0\alpha_1\alpha_2 \cdots$ it guesses a set of commands $R \subseteq \mathit{Comms}_Q$ that is a model of $\delta(q, \alpha_k)$. It then spawns one copy of itself for

α		c	l	c	r	r	c	c	l	r	l	l		$\cdots$
q	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	q_{11}		$\cdots$
γ				B				B	B					
		A	A	A	A		A	A	A	A	A	A		$\cdots$
	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$		

Figure 3 Run of a VPA $\mathfrak{A}$ on the word *clcrrcclrll*.

each command $(d, q, q') \in R$ and executes the command with that copy. If $d = \rightarrow_a$ and if α_k is a matched call, the copy jumps to the position of the matching return of α_k and transitions to state q'. Otherwise the automaton advances to position $k+1$ and transitions to state q. All copies of $\mathfrak{A}$ continue in parallel. A single copy of $\mathfrak{A}$ is successful if the highest color visited infinitely often is even. A 1-AJA accepts α if all of its copies are successful.

▶ **Lemma 7** ([4]). *For every 1-AJA $\mathfrak{A}$ there exists an effectively constructible equivalent BVPA $\mathfrak{A}_{vp}$ with $|\mathfrak{A}_{vp}| \in \mathcal{O}(2^{|\mathfrak{A}|})$.*

For a given VLDL formula φ we now inductively construct a 1-AJA that recognizes the same language as φ. The main difficulty lies in the translation of formulas of the form $\langle\mathfrak{A}\rangle\varphi$, since these require us to translate TVPAs over finite words into 1-AJAs over infinite words. We do so by adapting the idea for the translation from BVPAs to 1-AJAs from [4] and by combining it with the bottom-up translation from LDL into alternating automata in [6].

▶ **Lemma 8.** *For any VLDL formula φ there exists an effectively constructible equivalent 1-AJA $\mathfrak{A}_\varphi$ with $|\mathfrak{A}_\varphi| \in \mathcal{O}(|\varphi|^2)$.*

Proof. We construct the automaton inductively over the structure of φ. The case $\varphi = p$ is trivial. For Boolean operations, we obtain $\mathfrak{A}_\varphi$ by closure of 1-AJAs under these operations [4]. If $\varphi = [\mathfrak{A}]\varphi'$ we use the identity $[\mathfrak{A}]\varphi' \equiv \neg\langle\mathfrak{A}\rangle\neg\varphi'$ and construct $\mathfrak{A}_{\neg\langle\mathfrak{A}\rangle\neg\varphi'}$ instead.

We now consider $\varphi = \langle\mathfrak{A}\rangle\varphi'$, where $\mathfrak{A}$ is some TVPA and sketch the construction of $\mathfrak{A}_\varphi$. By induction we obtain a 1-AJA $\mathfrak{A}'$ equivalent to φ'. $\mathfrak{A}_\varphi$ simulates a run of $\mathfrak{A}$ on a prefix of α and, upon acceptance, nondeterministically transitions into $\mathfrak{A}'$.

Consider an initial run of $\mathfrak{A}$ on such a prefix w. Since w is finite, $steps(w)$ is finite as well. Hence, each stack height may only be encountered finitely often at a step. At the last visit to a step of a given height, $\mathfrak{A}$ either accepts, or it reads a call action. The symbol pushed onto the stack in that case does not influence the remainder of the run. Such a run on the word *clcrrcclrll* is shown in Figure 3, where c is a call, r is a return, and l is a local action.

The idea for the simulation of the run of $\mathfrak{A}$ is to have a main copy of $\mathfrak{A}_\varphi$ that jumps along the steps of the input word. When $\mathfrak{A}_\varphi$ encounters a call $c \in \Sigma_c$ it guesses whether or not $\mathfrak{A}$ encounters the current stack height again. If it does, then $\mathfrak{A}_\varphi$ guesses $q', q'' \in Q$ and $A \in \Gamma$ such that (q, c, q', A) is a transition of $\mathfrak{A}$, it jumps to the matching return of c in state q'' and spawns a copy that verifies that $\mathfrak{A}$ can go from the configuration (q', A) to the configuration $(q'', \bot)$. If $\mathfrak{A}$ never returns to the current stack height, then $\mathfrak{A}_\varphi$ only guesses $q' \in Q$ and $A \in \Gamma$ such that (q, c, q', A) is a transition of $\mathfrak{A}$, moves to state q', and stores in its state space that it may not read any returns anymore. This is repeated until the main copy guesses that $\mathfrak{A}'$ accepts.

The run of such a 1-AJA corresponding to the run of $\mathfrak{A}$ shown in Figure 3 is shown in Figure 4. The gray line indicates the stack height, while the solid and dashed black paths represent the run of the main automaton and of the verifying automata, respectively. Dotted lines indicate spawning a verifying automaton. For readability, the figure does not include copies of the automata that are spawned to verify that the tests of $\mathfrak{A}$ hold true. States of

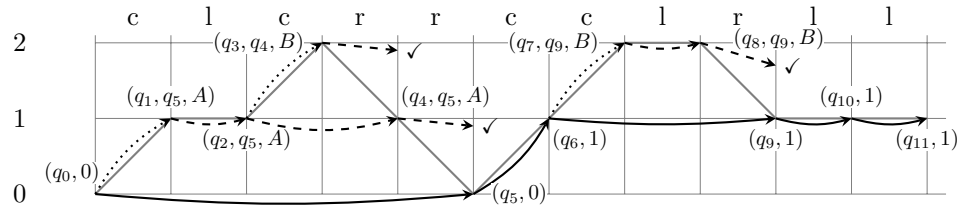

Figure 4 Behavior of 1-AJA on the word *clcrrcclrll*.

the form $(q, 0)$ denote the main copy of the automaton that has not yet ignored any call actions, while states of the form $(q, 1)$ denote copies that have done so. The states (q, q', A) denote verification copies that verify $\mathfrak{A}$'s capability to move from the configuration (q, A) to the configuration $(q', \bot)$. The verification automata work similarly to the main automaton, except that they assume that all pushed symbols to be eventually popped and reject if they encounter an unmatched call. Details can be found in the full version [12]. ◀

By combining Lemmas 7 and 8 we see that BVPAs are at least as expressive as VLDL formulas. This proves the direction from logic to automata of Theorem 3. The construction via 1-AJAs yields automata of exponential size in the number of states. This blowup is unavoidable, which can be shown by relying on the analogous lower bound for translating LTL into Büchi automata, obtained by encoding an exponentially bounded counter in LTL.

▶ **Lemma 9.** *There exists a pushdown alphabet $\widetilde{\Sigma}$ such that for all $n \in \mathbb{N}$ there exists a language L_n that is defined by a VLDL formula over $\widetilde{\Sigma}$ of polynomial size in n, but every BVPA over $\widetilde{\Sigma}$ recognizing L_n has at least exponentially many states in n.*

After having shown that VLDL has the same expressiveness as BVPAs, we now turn our attention to several decision problems for this logic. Namely, we study the satisfiability and the validity problem, as well as the model checking problem. Moreover, we consider the problem of solving visibly pushdown games with VLDL winning conditions.

5 Satisfiability and Validity are ExpTime-complete

We say that a VLDL formula φ is satisfiable if it has a model. Dually, we say that φ is valid if all words are models of φ. Instances of the satisfiability and validity problem consist of a VLDL formula φ and ask whether φ is satisfiable and valid, respectively. Both problems are decidable in exponential time. We also show both problems to be ExpTime-hard.

▶ **Theorem 10.** *The satisfiability and the validity problem for VLDL are* ExpTime*-complete.*

Proof. Due to duality, we only show ExpTime-completeness of the satisfiability problem. Membership follows from the 1-AJA-emptiness-problem being in ExpTime [4] and Lemma 8.

We show ExpTime-hardness by a polynomial-time reduction from the word problem for polynomially space-bounded alternating Turing machines. Our proof is based on the reduction of this problem to the problem of model checking pushdown systems against LTL specifications from the full version of [3]. In that reduction, a run of an alternating Turing machine is encoded as a pair of a pushdown system, which checks the general format of the encoding using its stack, and an LTL specification, which checks additional properties without using a stack. We adapt this proof by checking the properties asserted by the pushdown system with a visibly pushdown automaton. Also, we encode the specification of the general format in a VLDL formula. Technical details can be found in the full version [12]. ◀

6 Model Checking is ExpTime-complete

We now consider the model checking problem for VLDL. An instance of the model checking problem consists of a VPS $\mathcal{S}$, an initial state q_I of $\mathcal{S}$, and a VLDL formula φ and asks whether $\mathit{traces}(\mathcal{S}, q_I) \subseteq L(\varphi)$ holds true, where $\mathit{traces}(\mathcal{S}, q_I)$ denotes the set obtained by mapping each run of $\mathcal{S}$ starting in q_I to its sequence of labels. This problem is decidable in exponential time due to Lemma 8 and an exponential-time model checking algorithm for 1-AJAs [4]. Moreover, the problem is ExpTime-hard, as it subsumes the validity problem.

▶ **Theorem 11.** *Model checking VLDL specifications against VPS's is* ExpTime*-complete.*

Proof. Membership in ExpTime follows from Lemma 8 and the membership of the problem of checking visibly pushdown systems against 1-AJA specifications in ExpTime [4]. Moreover, since the validity problem for VLDL is ExpTime-hard and since validity of φ is equivalent to $\mathit{traces}(\mathcal{S}_{univ}) \subseteq \varphi$, where $\mathcal{S}_{univ}$ with $\mathit{traces}(\mathcal{S}_{univ}) = \Sigma^\omega$ is effectively constructible in constant time, the model checking problem for VLDL is ExpTime-hard as well. ◀

7 Solving VLDL Games is 3ExpTime-complete

In this section we investigate visibly pushdown games with winning conditions given by VLDL formulas. We consider games with two players, called Player 0 and Player 1, respectively.

A two-player game with VLDL winning condition $\mathcal{G} = (V_0, V_1, \Sigma, E, v_I, \ell, \varphi)$ consists of two disjoint, at most countably infinite sets V_0 and V_1 of vertices, where we define $V := V_0 \cup V_1$, a finite alphabet Σ, an initial state $v_I \in V$, a set of edges $E \subseteq V \times V$, a labeling $\ell\colon V \to \Sigma$, and a VLDL formula φ over some partition of Σ, called the winning condition.

A play $\pi = v_0 v_1 v_2 \cdots$ of $\mathcal{G}$ is an infinite sequence of vertices of $\mathcal{G}$ with $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. The play π is initial if $v_0 = v_I$. It is winning for Player 0 if $\ell(v_1)\ell(v_2)\ell(v_3)\cdots$ [2] is a model of φ. Otherwise π is winning for Player 1.

A strategy for Player i is a function $\sigma\colon V^*V_i \to V$, such that $(v, \sigma(w \cdot v)) \in E$ for all $v \in V_i$, $w \in V^*$. We call a play $\pi = v_0 v_1 v_2 \cdots$ consistent with σ if $\sigma(\pi') = v_{n+1}$ for all finite prefixes $\pi' = v_0 \cdots v_n$ of π with $v_n \in V_i$. A strategy σ is winning for Player i if all initial plays that are consistent with σ are winning for that player. We say that Player i wins $\mathcal{G}$ if she has a winning strategy. If either player wins $\mathcal{G}$, we say that $\mathcal{G}$ is determined.

A visibly pushdown game (VPG) with a VLDL winning condition $\mathcal{H} = (\mathcal{S}, Q_0, Q_1, q_I, \varphi)$ consists of a VPS $\mathcal{S} = (Q, \widetilde{\Sigma}, \Gamma, \Delta)$, a partition of Q into Q_0 and Q_1, an initial state $q_I \in Q$, and a VLDL formula φ over $\widetilde{\Sigma}$. The VPG $\mathcal{H}$ then defines the two-player game $\mathcal{G}_\mathcal{H} = (V_0, V_1, \Sigma, E, v_I, \ell, \varphi)$ with $V_i := Q_i \times ((\Gamma \setminus \{\bot\})^* \cdot \bot) \times \Sigma$, $v_I = (q_I, \bot, a)$ for some $a \in \Sigma$ (recall that the trace disregards the label of the initial vertex), $((q, \gamma, a), (q', \gamma', a')) \in E$ if there is an a'-labeled edge from (q, γ) to (q, γ') in the configuration graph $G_\mathcal{S}$, and $\ell\colon (q, \gamma, a) \mapsto a$. Solving a VPG $\mathcal{H}$ means deciding whether Player 0 wins $\mathcal{G}_\mathcal{H}$.

▶ **Proposition 12.** *VPGs with VLDL winning conditions are determined.*

Proof. Since each VLDL formula defines a language in ω-VPL due to Theorem 3, each VPG with VLDL winning condition is equivalent to a VPG with an ω-VPL winning condition. These are known to be determined [8]. ◀

[2] Note that the sequence of labels trace omits the label of the first vertex for technical reasons.

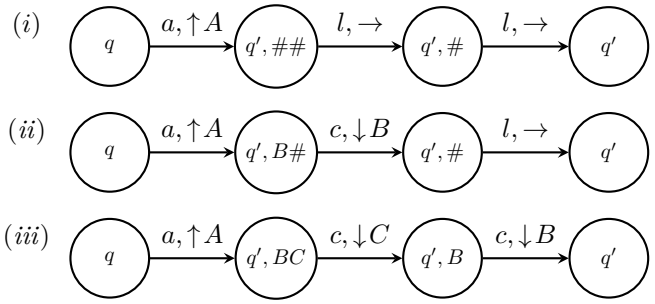

Figure 5 Construction of a VPG from a pushdown game for transitions of the forms (i) $(q, a, A, q', \varepsilon)$, (ii) (q, a, A, q', B), and (iii) (q, a, A, q', BC).

We show that solving VPGs with winning conditions specified in VLDL is harder than solving VPGs with winning conditions specified by BVPAs, i.e., they can be solved in triply exponential time. Moreover, they are complete for this complexity class.

▶ **Theorem 13.** *Solving VPGs with VLDL winning conditions is* 3ExpTime*-complete.*

Proof. We solve VPGs with VLDL winning conditions by constructing a BVPA $\mathfrak{A}_\varphi$ from the winning condition φ and by then solving the visibly pushdown game with a BVPA winning condition [8]. This approach takes triply-exponential time in $|\varphi|$ and exponential time in $|\mathcal{S}|$.

We show 3ExpTime-hardness of the problem by a reduction from solving pushdown games with LTL winning conditions, which is known to be 3ExpTime-complete [8]. Instead of, e.g., popping one symbol off the stack and pushing two others onto it, the resulting VPG splits these operations into individual pop- and push-operations, which are then carried out sequentially. The actions that still have to be carried out can be tracked using additional vertices. Since each stack operation can be split into at most three individual operations, this incurs only a linear blowup in the size of both the game and the winning condition.

A pushdown game with an LTL winning condition $\mathcal{H} = (\mathcal{S}, V_I, V_O, \psi)$ is defined similarly to a VPG, except for the relaxation that $\mathcal{S}$ may now be a traditional pushdown system instead of a visibly pushdown system. Specifically, we have $\Delta \subseteq (Q \times \Gamma \times \Sigma \times Q \times \Gamma^{\leq 2})$, where $\Gamma^{\leq 2}$ denotes the set of all words over Γ of at most two letters. Stack symbols are popped off the stack using transitions of the form $(q, A, a, q', \varepsilon)$, the top of the stack can be tested and changed with transitions of the form (q, A, a, q', B), and pushes are realized with transitions of the form (q, A, a, q', BC). Additionally, the winning condition is given as an LTL formula instead of a VLDL formula. The two-player game $\mathcal{G}_\mathcal{H}$ is defined analogously to the visibly pushdown game.

Since the pushdown game admits transitions such as (q, A, a, q', BC), which pop A off the stack and push B and C onto it, we need to split such transitions into several transitions in the visibly pushdown game. We modify the original game such that every transition of the original game is modeled by three transitions in the visibly pushdown game, up to two of which may be dummy actions that do not change the stack. As each transition may perform at most three operations on the stack, we can keep track of the list of changes still to be performed in the state space. We perform these actions using dummy letters c and l, which we add to Σ and read while performing the required actions on the stack. We choose the vertices $V'_X = V_X \cup (V_X \times (\Gamma \cup \{\#\})^{\leq 2})$ and the alphabet $\widetilde{\Sigma} = (\{c\}, \Sigma, \{l\})$.

We transform $\mathcal{H}$ as shown in Figure 5 and obtain the VPG $\mathcal{H}'$. Moreover, we transform the winning condition ψ of $\mathcal{H}$ into ψ' by inductively replacing each occurrence of $\mathbf{X}\psi$ by $\mathbf{X}^3\psi'$ and each occurrence of $\psi_1 \mathbf{U} \psi_2$ by $(\psi'_1 \vee c \vee l)\mathbf{U}(\psi'_2 \wedge \neg c \wedge \neg l)$. We subsequently translate

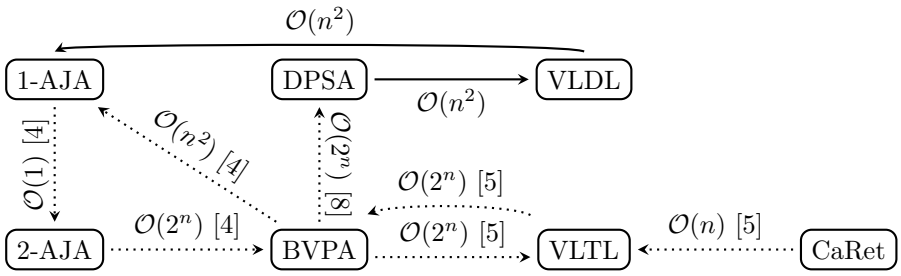

Figure 6 Formalisms capturing (subsets of) ω-VPL and translations between them.

the resulting LTL formula ψ' into an equivalent VLDL formula φ using Lemma 1. The input player wins $\mathcal{H}'$ with the winning condition φ if and only he wins $\mathcal{H}$ with the winning condition ψ. Hence, solving VPGs with VLDL winning conditions is 3ExpTime-hard. ◀

8 Conclusion

We have introduced Visibly Linear Dynamic Logic (VLDL) which strengthens Linear Dynamic Logic (LDL) by replacing the regular languages used as guards in the latter logic with visibly pushdown languages. VLDL captures precisely the class of ω-visibly pushdown languages. We have provided effective translations from VLDL to BVPA and vice versa with an exponential blowup in size in both directions. From automata to logic, this blowup cannot be avoided while it remains open whether or not it can be avoided in the other direction.

Figure 6 gives an overview over the known formalisms that capture ω-VPL and the translations between them. Our constructions are marked by solid lines, all others by dotted lines. All constructions are annotated with the blowup they incur.

In particular, there exist translations between VLTL and VLDL via BVPAs that incur a doubly-exponential blowup in both directions, as shown in Figure 6. In spite of this blowup the satisfiability problem and the model checking problem for both logics are ExpTime-complete. It remains open whether there exist efficient translations between the two logics.

We showed the satisfiability and the emptiness problem for VLDL, as well as model checking visibly pushdown systems against VLDL specifications, to be ExpTime-complete. Also, we proved that solving visibly pushdown games with VLDL winning conditions is 3ExpTime-complete.

Extending VLDL by replacing the guards with a more expressive family of languages quickly yields undecidable decision problems. In fact, using deterministic pushdown languages as guards already renders all decision problems discussed in this work undecidable [12].

In contrast to LDL [10] and VLTL [5], VLDL uses automata to define guards instead of regular or visibly rational expressions. We are currently investigating a variant of VLDL where the VPAs guarding the temporal operators are replaced by visibly rational expressions (with tests), which is closer in spirit to LDL.

Acknowledgments. The authors would like to thank Laura Bozzelli for providing the full version of [4] and Christof Löding for pointing out the 3ExpTime-hardness of solving infinite games for visibly pushdown games against LTL specifications.

References

1. Rajeev Alur, Kousha Ettesami, and Parthasarathy Madhusudan. A temporal logic of nested calls and returns. In *TACAS 2004*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.
2. Rajeev Alur and Parthasarathy Madhusudan. Visibly Pushdown Languages. In *STOC 2004*, pages 202–211. ACM, 2004.
3. Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR 1997*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997. Full version available at `http://www.liafa.univ-paris-diderot.fr/~abou/BEM97.pdf`.
4. Laura Bozzelli. Alternating Automata and a Temporal Fixpoint Calculus for Visibly Pushdown Languages. In L. Caires and V. T. Vasconcelos, editors, *CONCUR 2007*, volume 4703 of *LNCS*, pages 476–491. Springer, 2007.
5. Laura Bozzelli and César Sánchez. Visibly linear temporal logic. In *IJCAR 2014*, volume 8562 of *LNCS*, pages 418–483, 2014.
6. Peter Faymonville and Martin Zimmermann. Parametric Linear Dynamic Logic. *Inf. Comput.*, 2016. Article in press.
7. Martin Leucker and César Sanchéz. Regular linear temporal logic. In C. B. Jones, Z. Liu, and J. Woodcock, editors, *ICTAC 2007*, number 4711 in LNCS, pages 291–305, 2007.
8. Christof Löding, Parthasarathy Madhusudan, and Olivier Serre. Visibly Pushdown Games. In L. Lodaya and M. Mahajan, editors, *FSTTCS 2004*, volume 3328 of *LNCS*, pages 408–420. Springer, 2005.
9. Amir Pnueli. The temporal logic of programs. In *FOCS 1977*, pages 46–57. IEEE, 1977.
10. Moshe Vardi. The Rise and Fall of LTL. In G. D'Agostino and S. La Torre, editors, *EPTCS 54*, 2011.
11. Moshe Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. and Comp.*, 115:1–37, 1994.
12. Alexander Weinert and Martin Zimmermann. Visibly linear dynamic logic. *arXiv*, 1512.05177, 2015.
13. Pierre Wolper. Temporal logic can be more expressive. *Inf. and Cont.*, 56:72–99, 1983.